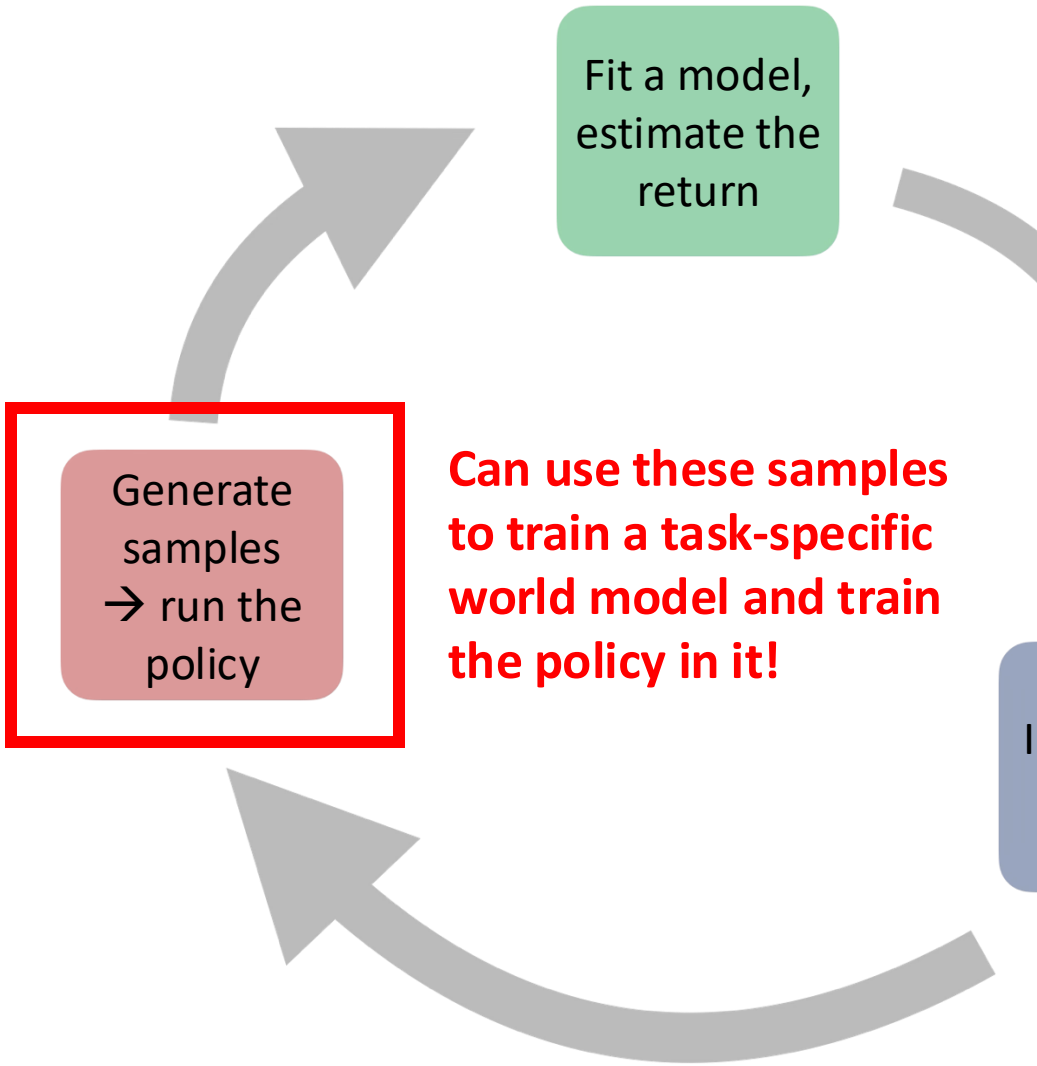
Policy Gradients and World Models

ESE 6510
Antonio Loquercio

Edouard Manet,

***A Bar at the Folies-Bergere*, 1882**

Policy Gradients in a Task-Specific World Model



evaluate returns

$$R_\tau = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$$

Can use these samples to train a task-specific world model and train the policy in it!

$$\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$$

This idea has a long history (Sutton, 1990)

Dyna, an Integrated Architecture for Learning, Planning, and Reacting

Richard S. Sutton
GTE Laboratories Incorporated
Waltham, MA 02254
sutton@gte.com

Abstract

Dyna is an AI architecture that integrates learning, planning, and reactive execution. Learning methods are used in Dyna both for compiling planning results and for updating a model of the effects of the agent's actions on the world. Planning is incremental and can use the probabilistic and ofttimes incorrect world models generated by learning processes. Execution is fully reactive in the sense that no planning intervenes between perception and action. Dyna relies on machine learning methods for learning from examples—these are among the basic building blocks making up the architecture—yet is not tied to any particular method. This paper briefly introduces Dyna and discusses its strengths and weaknesses with respect to other architectures.

1 Introduction to Dyna

The Dyna architecture attempts to integrate

- Trial-and-error learning of an optimal *reactive policy*, a mapping from situations to actions;
- Learning of domain knowledge in the form of an *action model*, a black box that takes as input a situation and action and outputs a prediction of the immediate next situation;
- Planning: finding the optimal reactive policy given domain knowledge (the action model);
- Reactive execution: No planning intervenes between perceiving a situation and responding to it.

Figure 1: The Problem Formulation Used in Dyna. The agent's object is to maximize the total reward it receives over time.[1]

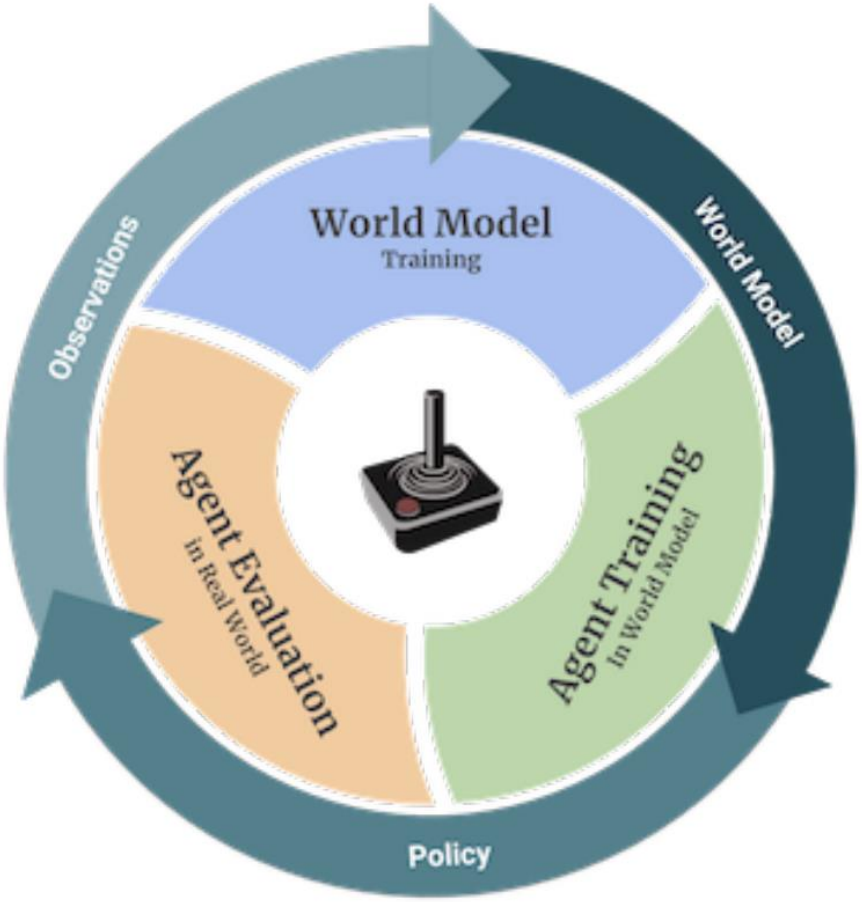
REPEAT FOREVER:

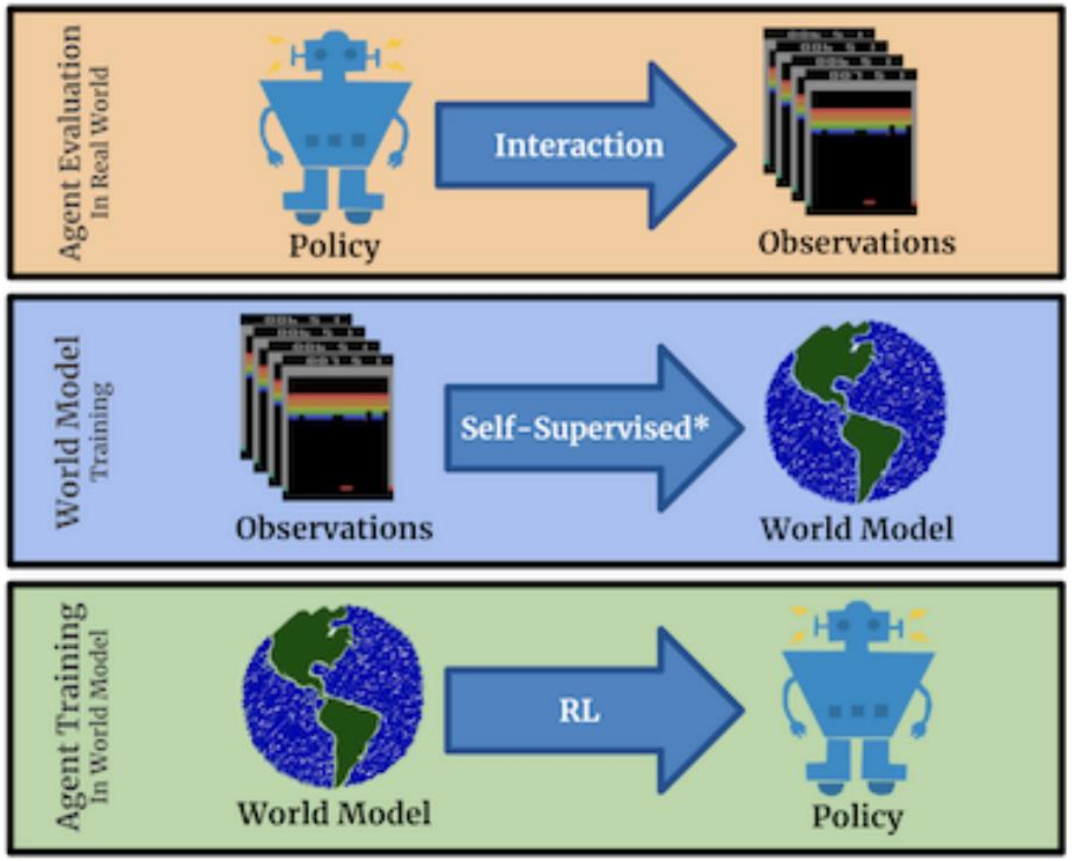
1. Observe the world's state and reactively choose an action based on it;
2. Observe resultant reward and new state;
3. Apply reinforcement learning to this experience;
4. Update action model based on this experience;
5. Repeat K times:
   - 5.1 Choose a hypothetical world state and action;
   - 5.2 Predict resultant reward and new state using action model;
   - 5.3 Apply reinforcement learning to this hypothetical experience.

Figure 2: A Generic Dyna Algorithm.

The main idea of Dyna is the old, commonsense idea that planning is 'trying things in your head,' using an internal model of the world (Craik, 1943; Dennett, 1978; Sutton &

PPO updates inside a world model



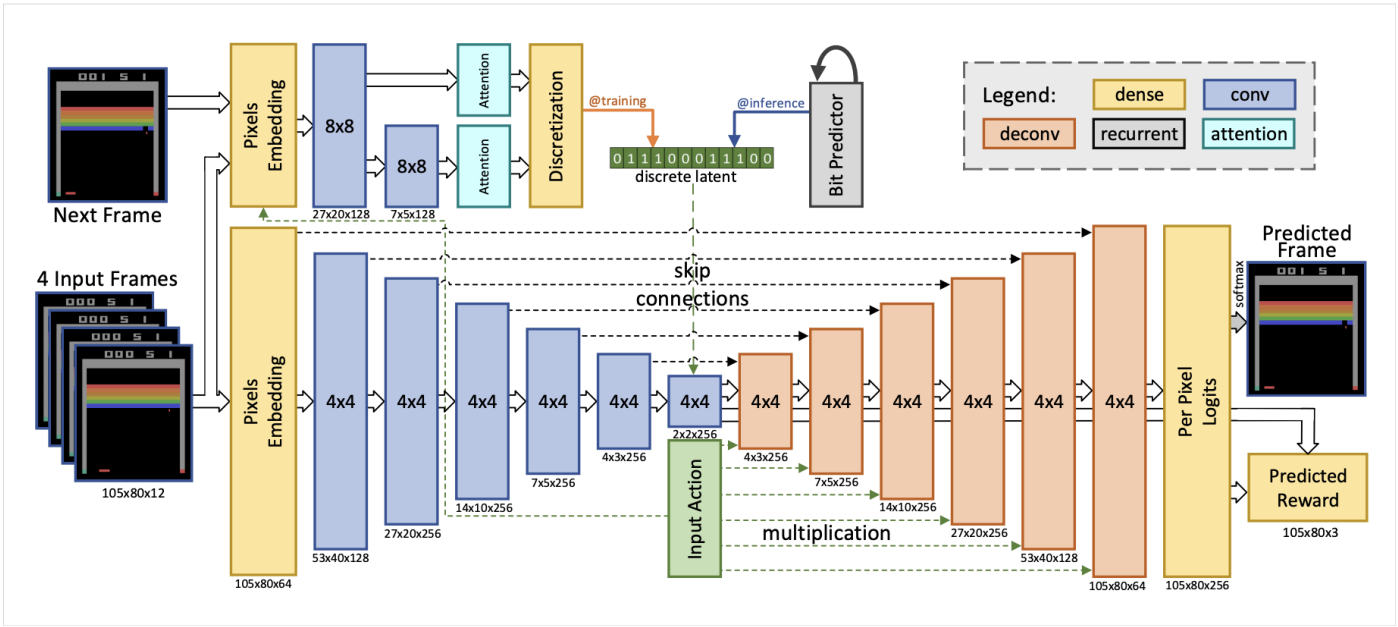


Model-based reinforcement learning for Atari, Kaiser et al., 2020

Simulated Policy Learning (SimPLe)

Algorithm 1: Pseudocode for SimPLe

Initialize policy π
Initialize model parameters θ of env'
Initialize empty set $\mathbf{D}$
while not done **do**
  ▷ collect observations from real env.
  $\mathbf{D} \leftarrow \mathbf{D} \cup \text{COLLECT}(env, \pi)$
  ▷ update model using collected data.
  $\theta \leftarrow \text{TRAIN\_SUPERVISED}(env', \mathbf{D})$
  ▷ update policy using world model.
  $\pi \leftarrow \text{TRAIN\_RL}(\pi, env')$
end while

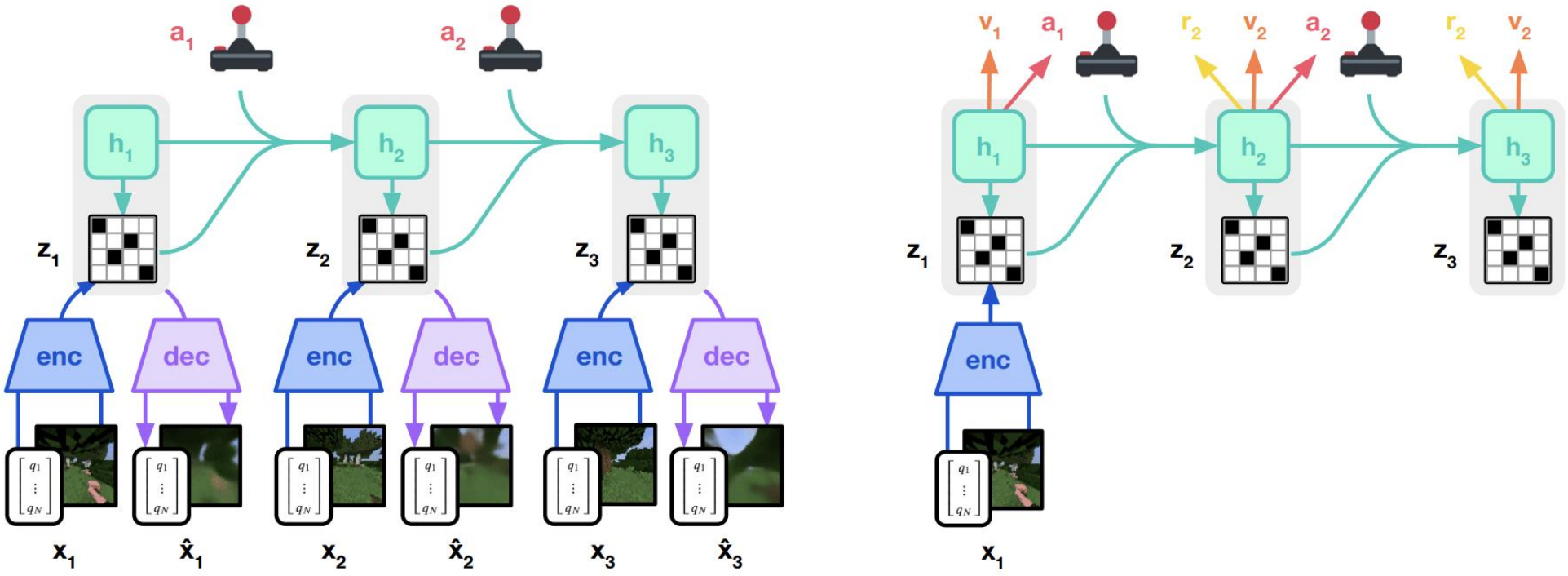


- Model is stochastic (VAE style), avoids blurring between modes of the distribution
- It uses a discrete latent (similar to VQ-VAE), which makes optimization harder but empirically gives better predictions.
- Do we really need a model in pixel space?

Model-based reinforcement learning for Atari, Kaiser et al., 2020

The Dreamer Saga (v1,v2,v3)

Learn a model in a latent space: No pixel-level predictions during planning.

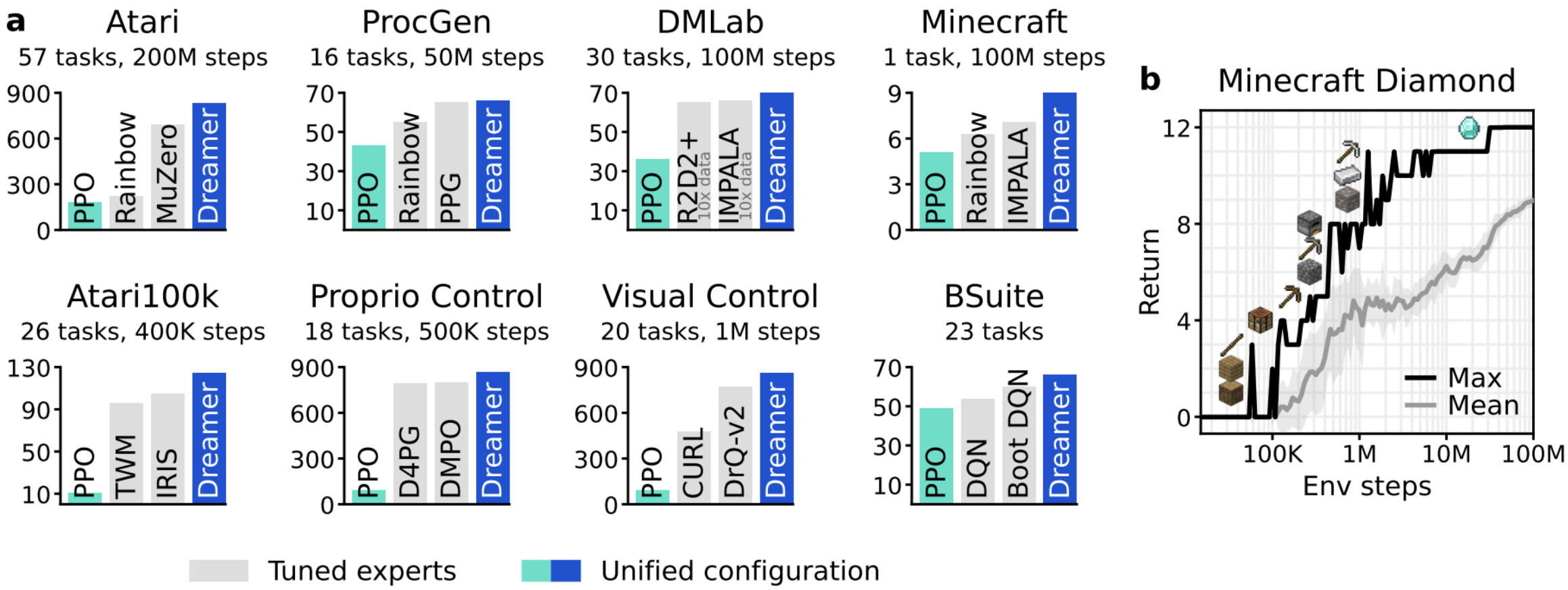


(a) World Model Learning

(b) Actor Critic Learning

Mastering Diverse Domains through World Models, Hafner et al., 2024

The Dreamer Saga (v1,v2,v3): Amazing Results!



Mastering Diverse Domains through World Models, Hafner et al., 2024

The Dreamer Saga (v1,v2,v3): Many moving parts

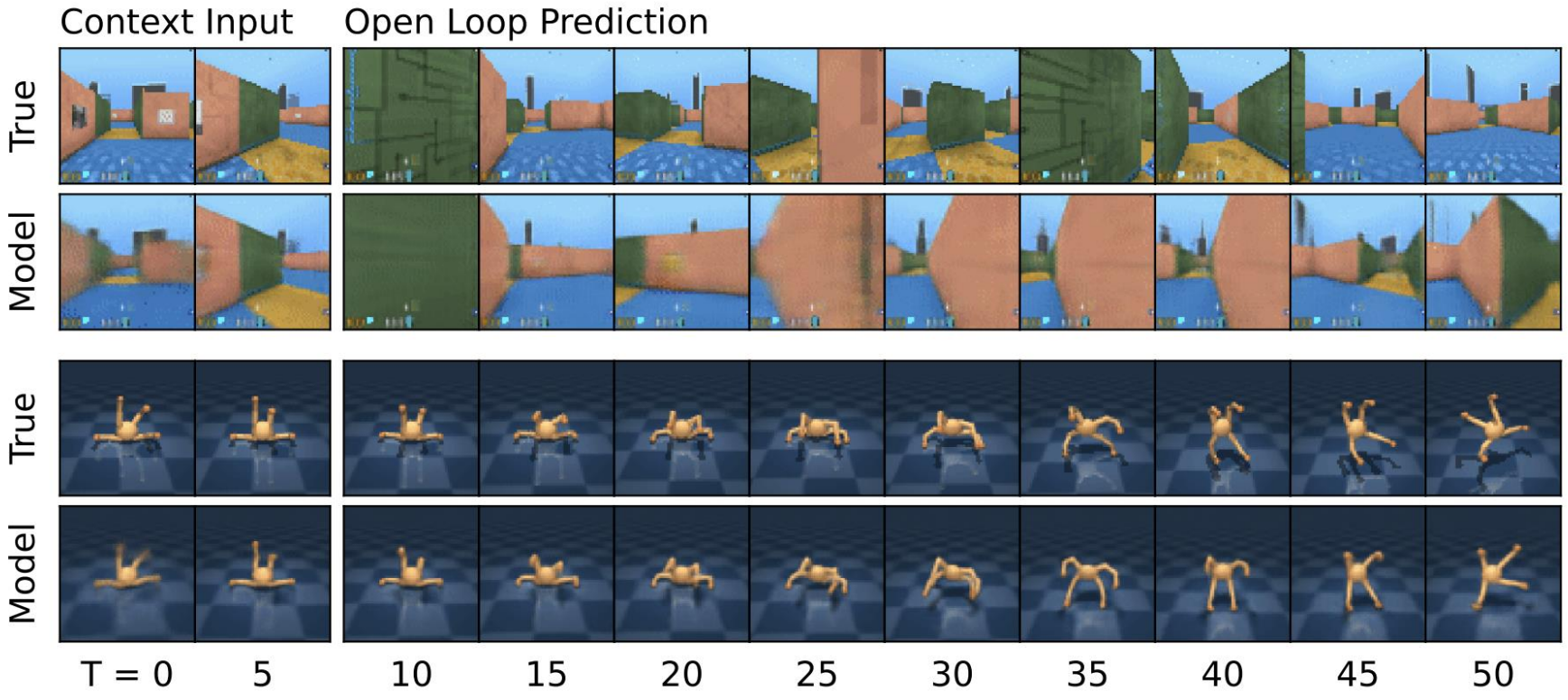
- Need several tricks to avoid catastrophic drifting of the model

$$
\text{RSSM} \begin{cases} \text{Sequence model:} & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\ \text{Encoder:} & z_t \sim q_\phi(z_t \mid h_t, x_t) \\ \text{Dynamics predictor:} & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\ \text{Reward predictor:} & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\ \text{Continue predictor:} & \hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t, z_t) \\ \text{Decoder:} & \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t) \end{cases}
$$

Mastering Diverse Domains through World Models, Hafner et al., 2024

The Dreamer Saga (v1,v2,v3): Model Drifts

- Can only forward propagate for a limited number of steps (T=16)
- Policy gradients suffer from limited rollout length, but learning a value function can help.



Mastering Diverse Domains through World Models, Hafner et al., 2024

The Dreamer Saga (v1,v2,v3)

- Works well for sparse reward problems with a discrete action space, especially when the state space is very large (images).
- However, it struggles with continuous action spaces. To solve the problem, Dreamer v1 and Dreamer v2 exploit the continuity and differentiability of the model and do direct backpropagations of rewards with respect to actions.

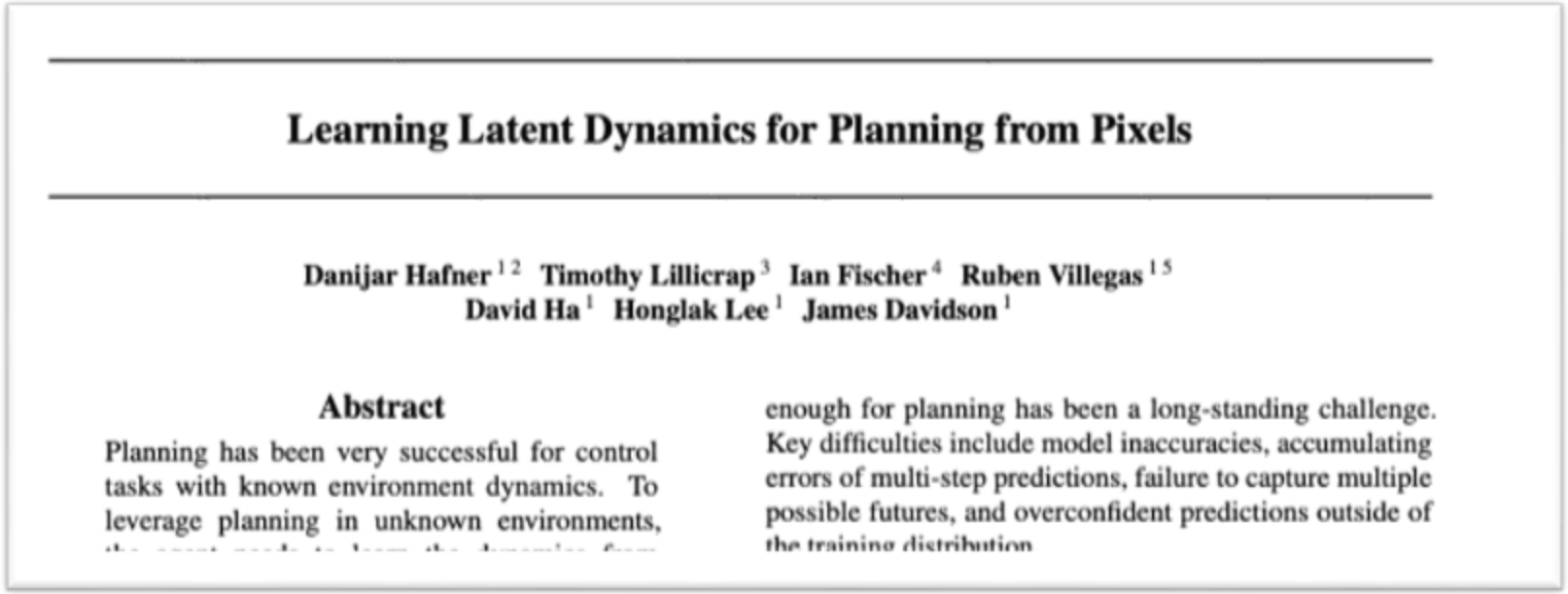
$$\mathcal{L}(\psi) \doteq \mathrm{E}_{p_\phi, p_\psi}\left[\sum_{t=1}^{H-1}\Big(\underbrace{-\rho \ln p_\psi(\hat{a}_t \mid \hat{z}_t)\,\mathrm{sg}(V_t^\lambda - v_\xi(\hat{z}_t))}_{\text{reinforce}} \underbrace{-(1-\rho)V_t^\lambda}_{\text{dynamics backprop}} \underbrace{-\eta\,\mathrm{H}[a_t|\hat{z}_t]}_{\text{entropy regularizer}}\Big)\right].$$

- Since we have a differentiable model, can't we do anything better than policy gradients?

Mastering Diverse Domains through World Models, Hafner et al., 2024

Planning in a latent world model

- Initially proposed in Dyna (Sutton, 1990). More recently, popularized by PlaNet (Hafner et al, 2018).
- Uses CEM to plan inside the latent model (MPC-Style)
- Works okayish (CEM is compute-intensive at test-time and has vanishing exploration problems)



Learning Latent Dynamics for Planning from Pixels

Danijar Hafner [1 2] Timothy Lillicrap [3] Ian Fischer [4] Ruben Villegas [1 5]
David Ha [1] Honglak Lee [1] James Davidson [1]

Abstract

Planning has been very successful for control tasks with known environment dynamics. To leverage planning in unknown environments, enough for planning has been a long-standing challenge. Key difficulties include model inaccuracies, accumulating errors of multi-step predictions, failure to capture multiple possible futures, and overconfident predictions outside of the training distribution.

Planning in a latent world model: TD-MPC (v1,v2) Saga

- Run MPC (more specifically, MPPI) inside the task-specific world model, using a prediction horizon H.
- Terminal cost approximated by a Q function. Trained together with the model from TD-targets.

$$\mu^*, \sigma^* = \arg\max_{(\mu,\sigma)} \mathop{\mathbb{E}}_{(\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_{t+H}) \sim \mathcal{N}(\mu, \sigma^2)} \left[\gamma^H Q(\mathbf{z}_{t+H}, \mathbf{a}_{t+H}) + \sum_{h=t}^{H-1} \gamma^h R(\mathbf{z}_h, \mathbf{a}_h) \right]$$

Terminal cost

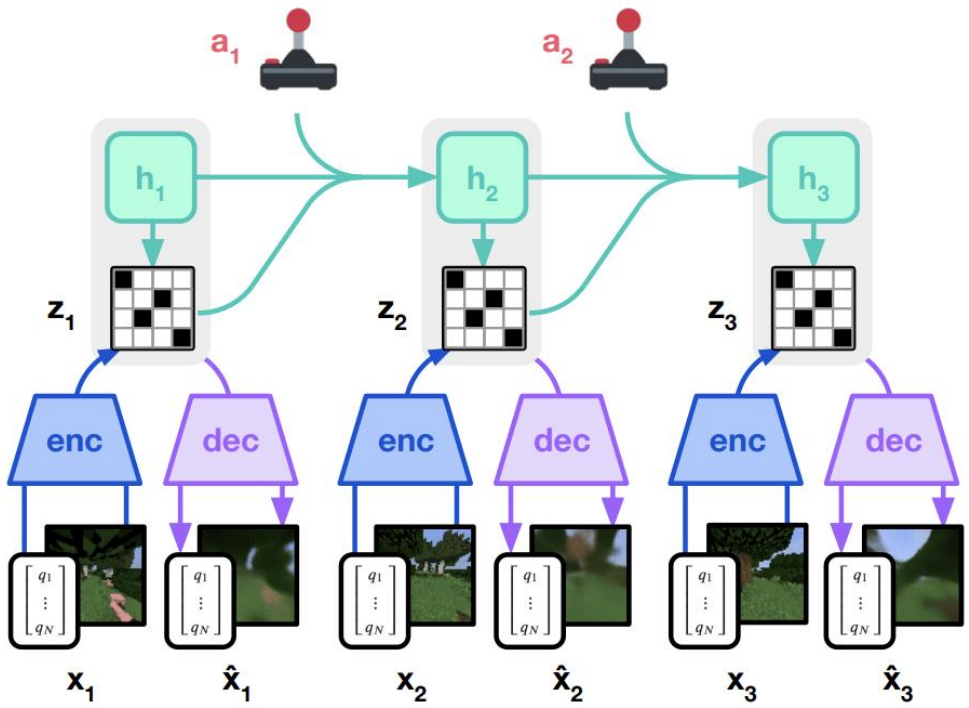
Stage cost

- Many tricks are required to make this work (model regularization, ensemble Q functions with EMA, warm-start with policy prior).

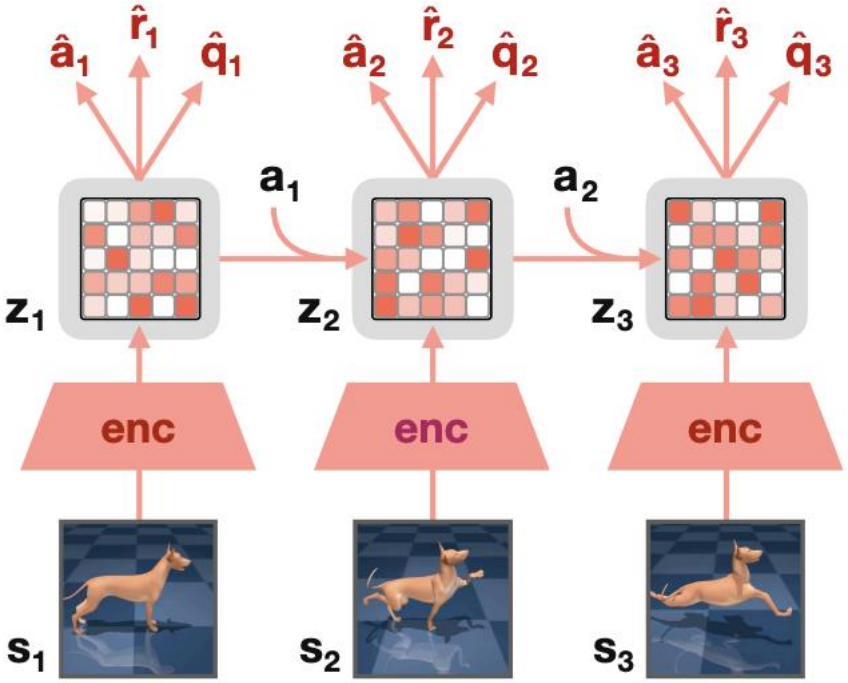
TD-MPC2: Scalable, Robust World Models for Continuous Control, Hansen et al, 2024

Planning in a latent world model: TD-MPC (v1,v2) Saga

Dreamer's Model



TD-MPC's Model



Do you notice any difference?

Planning in a latent world model: TD-MPC (v1,v2) Saga

- A neat finding: the model does not need to decode future pixel-level observations. Fewer moving parts.
- It makes the optimization easier, increasing the focus of the optimizer towards a useful world model rather than an accurate world model.
- Unclear if that's an advantage for new tasks (particularly at scale).

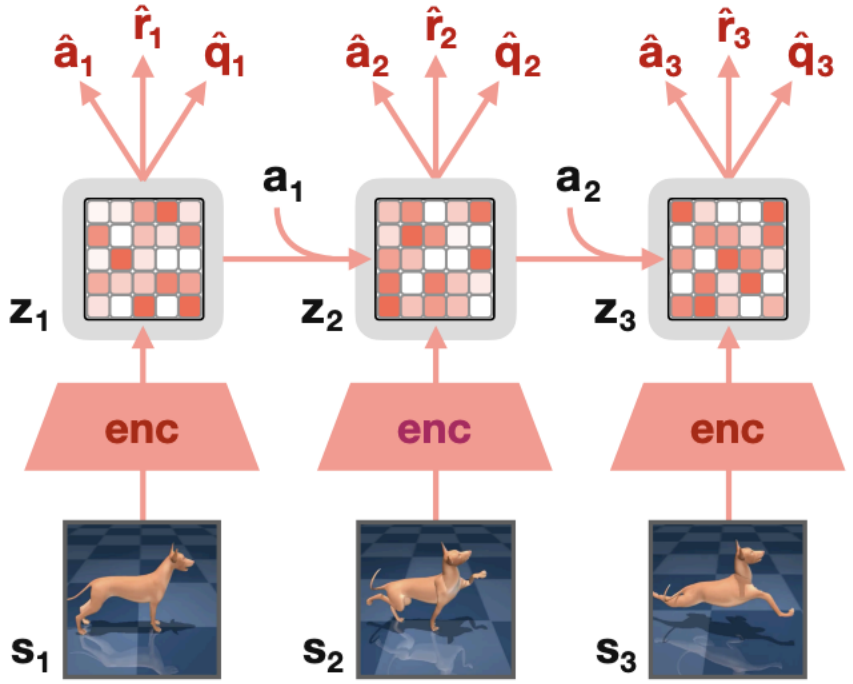
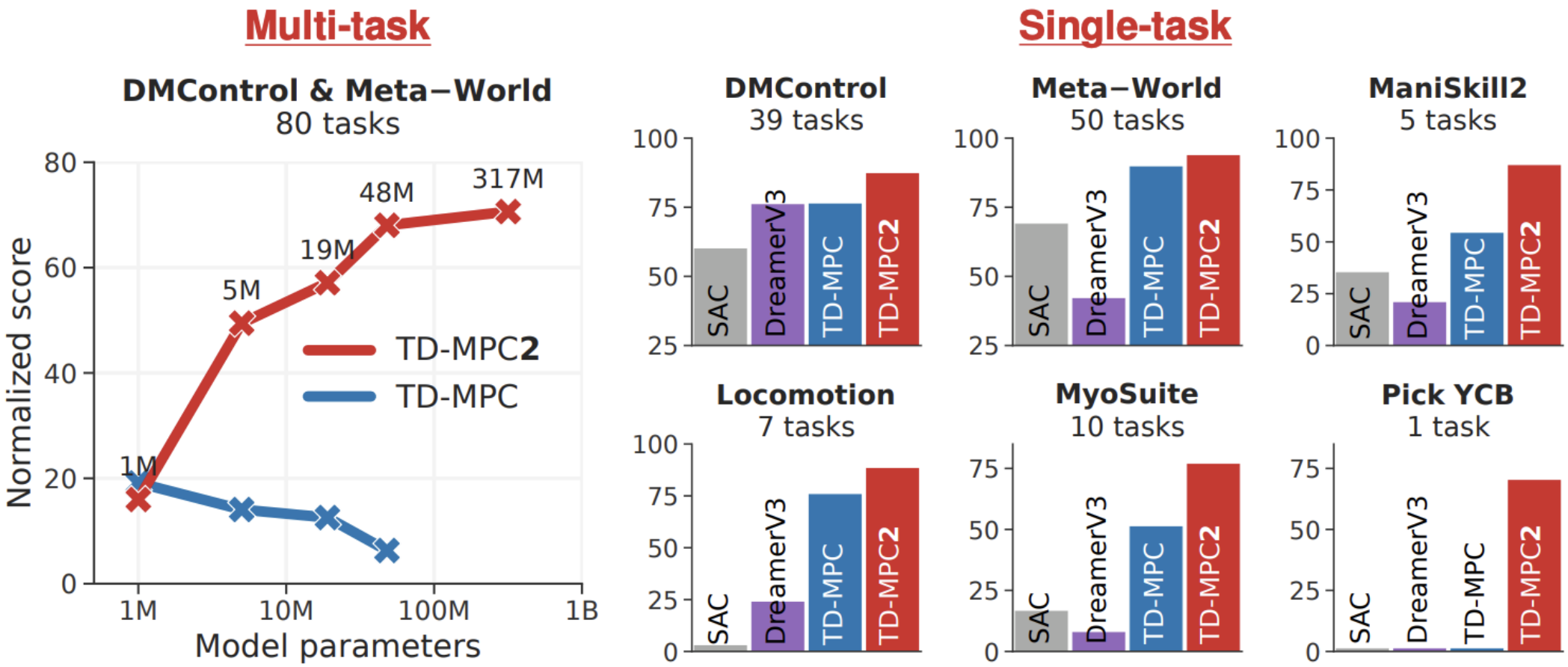


Figure 3. **The TD-MPC2 architecture.** Observations $\mathbf{s}$ are encoded into their (normalized) latent representation $\mathbf{z}$. The model then recurrently predicts actions $\hat{\mathbf{a}}$, rewards $\hat{r}$, and terminal values $\hat{q}$, *without* decoding future observations.

TD-MPC2: Scalable, Robust World Models for Continuous Control, Hansen et al, 2024

Planning in a latent world model: TD-MPC (v1,v2) Saga

Much better than Dreamer and other baselines in continuous control problems!



TD-MPC2: Scalable, Robust World Models for Continuous Control, Hansen et al, 2024

Fresh off the market: Dreamer v4

Google DeepMind

Training Agents Inside of Scalable World Models

Danijar Hafner* Wilson Yan* Timothy Lillicrap

World models learn general knowledge from videos and simulate experience for training behaviors in imagination, offering a path towards intelligent agents. However, previous world models have been unable to accurately predict object interactions in complex environments. We introduce Dreamer 4, a scalable agent that learns to solve control tasks by reinforcement learning inside of a fast and accurate world model. In the complex video game Minecraft, the world model accurately predicts object interactions and game mechanics, outperforming previous world models by a large margin. The world model achieves real-time interactive inference on a single GPU through a shortcut forcing objective and an efficient transformer architecture. Moreover, the world model learns general action conditioning from only a small amount of data, allowing it to extract the majority of its knowledge from diverse unlabeled videos. We propose the challenge of obtaining diamonds in Minecraft from only offline data, aligning with practical applications such as robotics where learning from environment interaction can be unsafe and slow. This task requires choosing sequences of over 20,000 mouse and keyboard actions from raw pixels. By learning behaviors in imagination, Dreamer 4 is the first agent to obtain diamonds in Minecraft purely from offline data, without environment interaction. Our work provides a scalable recipe for imagination training, marking a step towards intelligent agents.

Dreamer v4

- No interaction with the environment whatsoever
- Instead, it leverages a large dataset of offline demonstrations (videos + actions + rewards).
- General idea:

Algorithm 1 Dreamer 4

Phase 1: World Model Pretraining

- Train tokenizer on videos using (5).
- Train world model on tokenized videos and optionally actions using (7).

Phase 2: Agent Finetuning

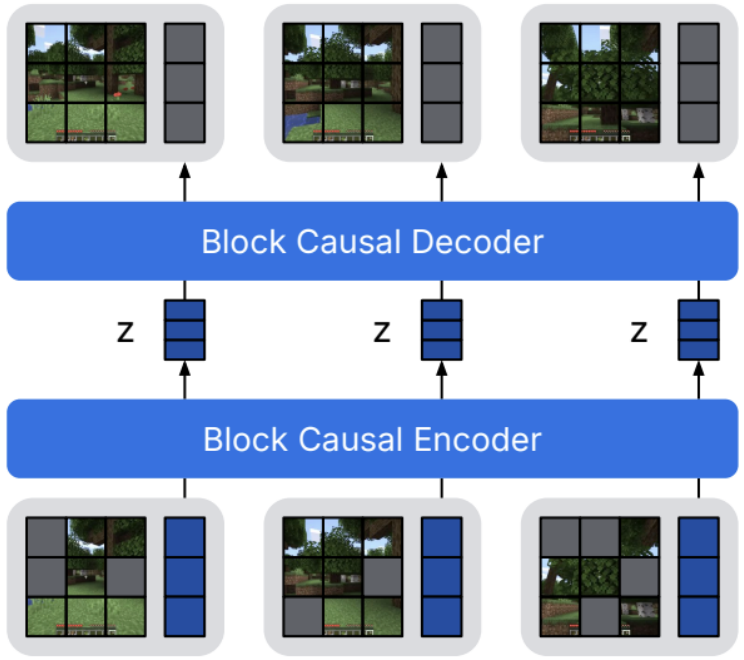
- Finetune world model with task inputs for policy and reward heads using (7) and (9).

Phase 3: Imagination Training

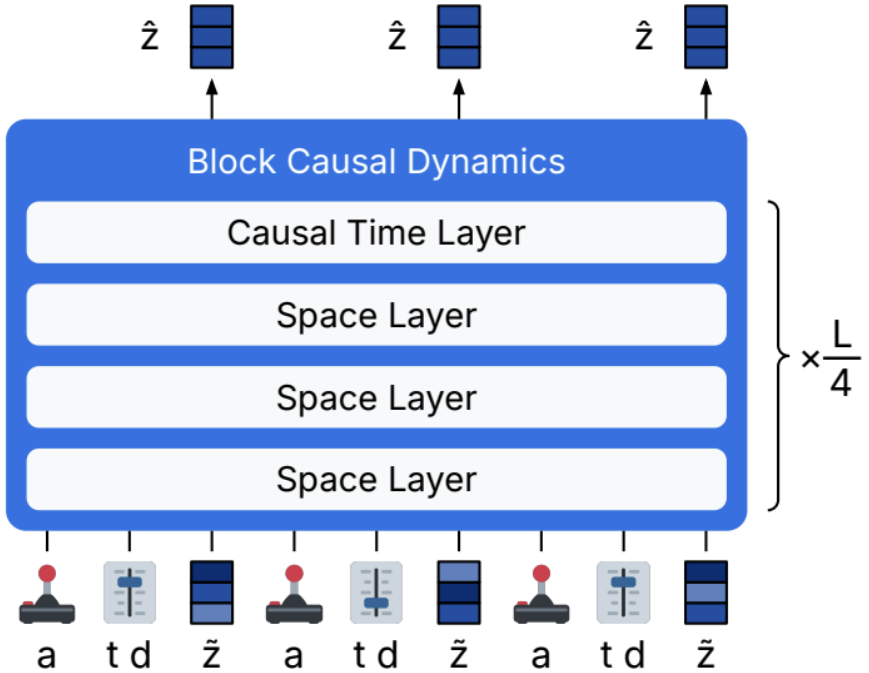
- Optimize policy head using (11) and value head using (10) on trajectories generated by the world model and the policy head.

Dreamer v4

- Very fancy video and dynamics prediction model



(a) Causal Tokenizer



(b) Interactive Dynamics

Dreamer v4

- Video model is a shortcut flow-matching model (much faster at inference time than traditional diffusion models).
- Trained with interleaved actions and observations. Technically, you don’t need actions for every sequence.
- Learn a policy prior and reward model from the representations of the video model.
- Finetune agent only in the world model using PMPO (only sign of advantage used in the update):

$$\mathcal{L}(\theta) = \frac{1-\alpha}{|\mathcal{D}^-|} \sum_{i \in \mathcal{D}^-} \ln \pi_\theta(a_i \mid s_i) - \frac{\alpha}{|\mathcal{D}^+|} \sum_{i \in \mathcal{D}^+} \ln \pi_\theta(a_i \mid s_i) + \frac{\beta}{N} \sum_{i=1}^{N} \mathrm{KL}[\pi_\theta(a_i \mid s_i) \,\|\, \pi_{\text{prior}}]$$

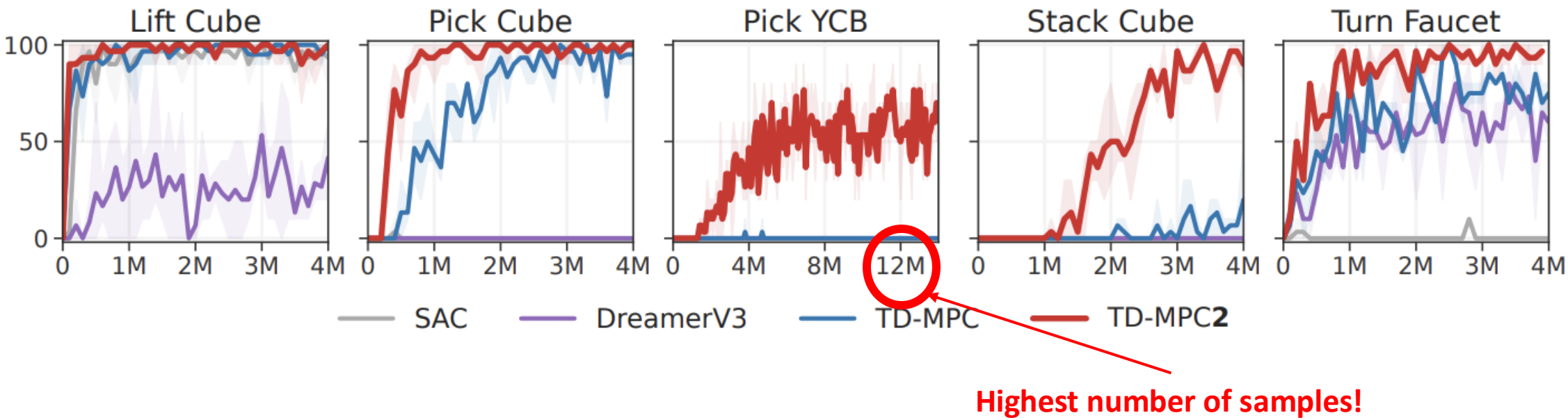
Summary

- Doing optimization inside a task-specific world model reduces the number of samples you need to collect by running the current policy in the environment.

- Effectively squeezing more out of each sample.

- When is this useful?
  - Slow/Dangerous to run the policy in the environment
  - Rewards are sparse (Why does direct policy gradient struggle?).
    - If the sum of rewards is often zero, the estimated values will be zero, and updates will be noisy. If there is a lucky rollout with a non-zero reward, direct policy learning throws it away immediately. Model-based learning can use it better.

However…



- If you can wait long enough (billions and not millions of samples) and you are careful about decaying exploration, PPO-style approaches tend to do as good or better than model-based approaches.

However…

- If I am anyway training in a simulator, I can play other tricks to make PPO-style approaches faster.

- Can you give some examples?

  - Dense reward shaping (foot contact, reaching rewards, etc.). Especially useful if I know more or less how the task should be solved.

  - Smart reset strategies/environment design:
    - Initialize the agent close to places where it can easily get rewards (e.g., close to gates in drone racing)
    - Stop the rollout if the agent is in a state where recovery is difficult (e.g., crashed)

Wait...

- With task-specific models, I can potentially learn **w/o a simulator**!

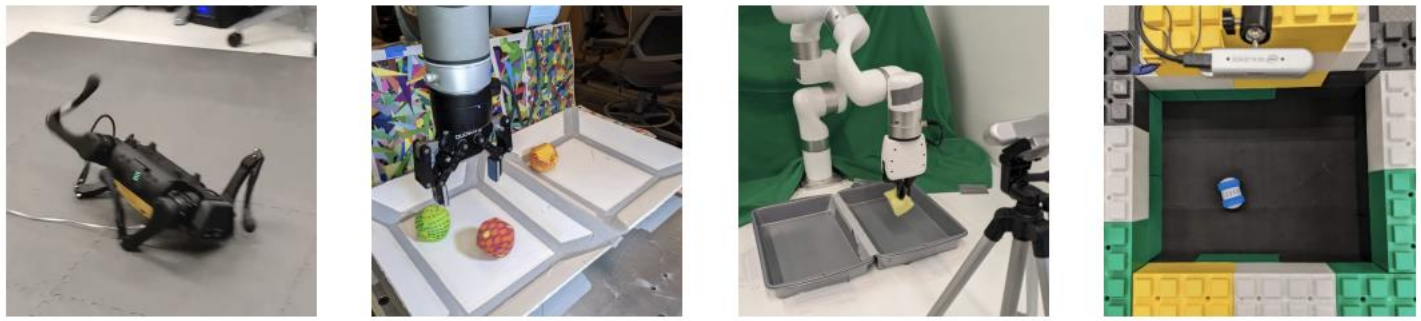
DayDreamer: World Models for Physical Robot Learning

Philipp Wu* **Alejandro Escontrela*** **Danijar Hafner***

Ken Goldberg **Pieter Abbeel**

University of California, Berkeley

*Equal contribution



(a) A1 Quadruped Walking (b) UR5 Visual Pick Place (c) XArm Visual Pick Place (d) Sphero Navigation

However…

- It does not seem to work well when observation is high-dimensional.
- It struggles beyond simple tasks where the robot can be easily reset.

However…

- It does not seem to work well when observation is high-dimensional.
- It struggles beyond simple tasks where the robot can be easily reset.

- We seem to live in a world where good simulation engineering and reward shaping work much better than learning the model from real-world robot interactions.
  - Unclear whether that's general or an artifact of the family of tasks we are currently very successful at (quadcopter flight, locomotion, pick-and-place).
  - Unclear whether powerful video models will change this (only cover one modality, potentially slower than real-time)
  - We are still far from a solid algorithmic solution to this problem! (One possible solution might be to have robots that don't break easily and can be run for long)